Code signing

By Contributors to Wikimedia projects

Published: 2006-01-19 · Archived: 2026-04-05 17:13:36 UTC

From Wikipedia, the free encyclopedia

Code signing is the process of digitally signing executables and scripts to confirm the software author and guarantee that the code has not been altered or corrupted since it was signed. The process employs the use of a cryptographic hash to validate authenticity and integrity.[1] Code signing was invented in 1995 by Michael Doyle, as part of the Eolas WebWish browser plug-in, which enabled the use of public-key cryptography to sign downloadable Web app program code using a secret key, so the plug-in code interpreter could then use the corresponding public key to authenticate the code before allowing it access to the code interpreter's APIs.[2][3]

Code signing can provide several valuable features. The most common use of code signing is to provide security when deploying; in some programming languages, it can also be used to help prevent namespace conflicts. Almost every code signing implementation will provide some sort of digital signature mechanism to verify the identity of the author or build system, and a checksum to verify that the object has not been modified. It can also be used to provide versioning information about an object or to store other metadata about an object.[4]

The efficacy of code signing as an authentication mechanism for software depends on the security of underpinning signing keys. As with other public key infrastructure (PKI) technologies, the integrity of the system relies on publishers securing their private keys against unauthorized access. Keys stored in software on general-purpose computers are susceptible to compromise. Therefore, it is more secure, and best practice, to store keys in secure, tamper-proof, cryptographic hardware devices known as hardware security modules or HSMs.[5]

Many code signing implementations will provide a way to sign the code using a system involving a pair of keys, one public and one private, similar to the process employed by TLS or SSH. For example, in the case of .NET, the developer uses a private key to sign their libraries or executables each time they build. This key will be unique to a developer or group or sometimes per application or object. The developer can either generate this key on their own or obtain one from a trusted certificate authority (CA).[6]

Code signing is particularly valuable in distributed environments, where the source of a given piece of code may not be immediately evident - for example Java applets, ActiveX controls and other active web and browser scripting code. Another important usage is to safely provide updates and patches to existing software.[7] Windows, Mac OS X, and most Linux distributions provide updates using code signing to ensure that it is not possible for others to maliciously distribute code via the patch system. It allows the receiving operating system to verify that the update is legitimate, even if the update was delivered by third parties or physical media (disks).[8]

Code signing is used on Windows and Mac OS X to authenticate software on first run, ensuring that the software has not been maliciously tampered with by a third-party distributor or download site. This form of code signing is not used on Linux because of that platform's decentralized nature, the package manager being the predominant

mode of distribution for all forms of software (not just updates and patches), as well as the open-source model allowing direct inspection of the source code if desired. Debian-based Linux distributions (among others) validate downloaded packages using public key cryptography.[9]

Trusted identification using a certificate authority (CA)

[edit]

The public key used to authenticate the code signature should be traceable back to a trusted root authority CA, preferably using a secure public key infrastructure (PKI). This does not ensure that the code itself can be trusted, only that it comes from the stated source (or more explicitly, from a particular private key).[10] A CA provides a root trust level and is able to assign trust to others by proxy. If a user trusts a CA, then the user can presumably trust the legitimacy of code that is signed with a key generated by that CA or one of its proxies. Many operating systems and frameworks contain built-in trust for one or more certification authorities. It is also commonplace for large organizations to implement a private CA, internal to the organization, which provides the same features as public CAs, but it is only trusted within the organization.

Extended validation (EV) code signing

[edit]

Extended validation (EV) code signing certificates are subject to additional validation and technical requirements. These guidelines are based on the CA/B Forum's Baseline Requirements and Extended Validation Guidelines. In addition to validation requirements specific to EV, the EV code signing guidelines stipulate that "the Subscriber's private key is generated, stored and used in a crypto module that meets or exceeds the requirements of FIPS 140-2 level 2."[11]

Certain applications, such as signing Windows 10 kernel-mode drivers, require an EV code signing certificate.[12] Additionally, Microsoft's IEBlog states that Windows programs "signed by an EV code signing certificate can immediately establish reputation with SmartScreen reputation services even if no prior reputation exists for that file or publisher."[13]

Sample EV code signing certificate

[edit]

This is an example of a decoded EV code signing certificate used by SSL.com to sign software. `SSL.com EV Code Signing Intermediate CA RSA R3` is shown as the Issuer's commonName, identifying this as an EV code signing certificate. The certificate's `Subject` field describes SSL Corp as an organization. `Code Signing` is shown as the sole X509v3 Extended Key Usage.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
```

```
            59:4e:2d:88:5a:2c:b0:1a:5e:d6:4c:7b:df:35:59:7d
    Signature Algorithm: sha256WithRSAEncryption
        Issuer:
            commonName                = SSL.com EV Code Signing Intermediate CA RSA R3
            organizationName          = SSL Corp
            localityName              = Houston
            stateOrProvinceName       = Texas
            countryName               = US
        Validity
            Not Before: Aug 30 20:29:13 2019 GMT
            Not After : Nov 12 20:29:13 2022 GMT
        Subject:
            1.3.6.1.4.1.311.60.2.1.3 = US
            1.3.6.1.4.1.311.60.2.1.2 = Nevada
            streetAddress             = 3100 Richmond Ave Ste 503
            businessCategory          = Private Organization
            postalCode                = 77098
            commonName                = SSL Corp
            serialNumber              = NV20081614243
            organizationName          = SSL Corp
            localityName              = Houston
            stateOrProvinceName       = Texas
            countryName               = US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c3:e9:ae:be:d7:a2:6f:2f:24 ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:36:BD:49:FF:31:2C:EB:AF:6A:40:FE:99:C0:16:ED:BA:FC:48:DD:5F

            Authority Information Access:
                CA Issuers - URI:http://www.ssl.com/repository/SSLcom-SubCA-EV-CodeSigning-RSA-4096-
                OCSP - URI:http://ocsps.ssl.com

            X509v3 Certificate Policies:
                Policy: 2.23.140.1.3
                Policy: 1.2.616.1.113527.2.5.1.7
                Policy: 1.3.6.1.4.1.38064.1.3.3.2
                  CPS: https://www.ssl.com/repository

            X509v3 Extended Key Usage:
                Code Signing
            X509v3 CRL Distribution Points:
```

```
                Full Name:
                  URI:http://crls.ssl.com/SSLcom-SubCA-EV-CodeSigning-RSA-4096-R3.crl

            X509v3 Subject Key Identifier:
                EC:6A:64:06:26:A7:7A:69:E8:CC:06:D5:6F:FA:E1:C2:9A:29:79:DE
            X509v3 Key Usage: critical
                Digital Signature
    Signature Algorithm: sha256WithRSAEncryption
         17:d7:a1:26:58:31:14:2b:9f:3b ...
```

The other model is the trust on first use model, in which developers can choose to provide their own self-generated key. In this scenario, the user would normally have to obtain the public key in some fashion directly from the developer to verify the object is from them for the first time. Many code signing systems will store the public key inside the signature. Some software frameworks and OSs that check the code's signature before executing will allow you to choose to trust that developer from that point on after the first run. An application developer can provide a similar system by including the public keys with the installer. The key can then be used to ensure that any subsequent objects that need to run, such as upgrades, plugins, or another application, are all verified as coming from that same developer.

Time-stamping was designed to circumvent the trust warning that will appear in the case of an expired certificate. In effect, time-stamping extends the code trust beyond the validity period of a certificate.[14]

In the event that a certificate has to be revoked due to a compromise, a specific date and time of the compromising event will become part of the revocation record. In this case, time-stamping helps establish whether the code was signed before or after the certificate was compromised.[14]

Developers need to sign their iOS and tvOS apps before running them on any real device and before uploading them to the App Store. This is needed to prove that the developer owns a valid Apple Developer ID. An application needs a valid profile or certificate so that it can run on the devices.[15]

Like any security measure, code signing can be defeated. Users can be tricked into running unsigned code, or even into running code that refuses to validate, and the system only remains secure as long as the private key remains private.[16][17]

It is also important to note that code signing does not protect the end user from any malicious activity or unintentional software bugs by the software author — it merely ensures that the software has not been modified by anyone other than the author. Sometimes, sandbox systems do not accept certificates, because of a false time-stamp or because of an excess usage of RAM.

Microsoft implements a form of code signing (based on Authenticode) provided for Microsoft tested drivers. Since drivers run in the kernel, they can destabilize the system or open the system to security holes. For this reason, Microsoft tests drivers submitted to its WHQL program. After the driver has passed, Microsoft signs that version of the driver as being safe. On 32-bit systems only, installing drivers that are not validated with Microsoft is possible after agreeing to allow the installation at a prompt warning the user that the code is unsigned. For .NET (managed) code, there is an additional mechanism called Strong Name Signing that uses Public/Private keys and

SHA-1 hash as opposed to certificates. However, Microsoft discourages reliance on Strong Name Signing as a replacement for Authenticode.[18]

The Code Signing Working Group of the CA/Browser Forum decided that starting June 1, 2023, all code signing certificates (not only the EA ones) should mandate private key storage on a physical media, such as in a hardware crypto module conforming to at least FIPS 140-2 Level 2 or Common Criteria EAL 4+.[19] The CAs subsequently issued announcements on compliance with the decision.[20][21][22][23][24][25][26]

Unsigned code in gaming and consumer devices

[edit]

In the context of consumer devices such as games consoles, the term "unsigned code" is often used to refer to an application which has not been signed with the cryptographic key normally required for software to be accepted and executed. Most console games have to be signed with a secret key designed by the console maker or the game will not load on the console (both to enforce Vendor lock-in and combat software piracy). There are several methods to get unsigned code to execute which include software exploits, the use of a modchip, a technique known as the swap trick or running a softmod.

It may not initially seem obvious why simply copying a signed application onto another DVD does not allow it to boot. On the Xbox, the reason for this is that the Xbox executable file (XBE) contains a media-type flag, which specifies the type of media that the XBE is bootable from. On nearly all Xbox software, this is set such that the executable will only boot from factory-produced discs, so simply copying the executable to burnable media is enough to stop the execution of the software.

However, since the executable is signed, simply changing the value of the flag is not possible as this alters the signature of the executable, causing it to fail validation when checked.

- Digital signature
- iOS jailbreaking
- PlayStation Portable homebrew
- Privilege escalation
- Rooting (Android OS)
- Symbian OS Security bypass

1. ^ *"Introduction to Code Signing (Windows)"*. *learn.microsoft.com. August 15, 2017. Archived from the original on February 6, 2024. Retrieved March 13, 2024.*
2. ^ *"WebWish: Our Wish is Your Command"*.
3. ^ Schroeder, H and Doyle, M. “Interactive Web Applications with Tcl/Tk”. Academic Professional, Boston, 1998, p. 14, ISBN 0122215400.
4. ^ *Hendric, William (2015). "A Complete overview of Trusted Certificates - CABForum" (PDF). Archived (PDF) from the original on 2019-04-22. Retrieved 2015-02-26.*
5. ^ *"Securing your Private Keys as Best Practice for Code Signing Certificates" (PDF).*

6. ^ *Hendric, William (17 June 2011). "What is Code Signing?". Archived from the original on 20 June 2018. Retrieved 26 February 2015.*
7. ^ *"Digital Signatures and Windows Installer - Win32 apps". learn.microsoft.com. January 7, 2021. Archived from the original on January 30, 2024. Retrieved March 13, 2024.*
8. ^ *windows-driver-content (2022-05-18). "Windows Secure Boot Key Creation and Management Guidance". learn.microsoft.com. Archived from the original on 2023-10-30. Retrieved 2023-09-22.*
9. ^ *"SecureApt - Debian Wiki". wiki.debian.org. Archived from the original on 2019-05-07. Retrieved 2019-05-07.*
10. ^ *"Code signing" (PDF). 2014-02-26. Archived (PDF) from the original on 2014-02-26. Retrieved 2014-02-21.*
11. ^ *"Guidelines For The Issuance And Management Of Extended Validation Code Signing Certificates" (PDF). CA/Browser Forum. Archived (PDF) from the original on 27 November 2019. Retrieved 4 December 2019.*
12. ^ *"Driver Signing Policy". Microsoft. Archived from the original on 9 December 2019. Retrieved 9 December 2019.*
13. ^ *"Microsoft SmartScreen & Extended Validation (EV) Code Signing Certificates". Microsoft. 14 August 2012. Archived from the original on 9 December 2019. Retrieved 9 December 2019.*
14. ^ Jump up to: ***a*** ***b*** *Morton, Bruce. "Code Signing" (PDF). CASC. Archived (PDF) from the original on 26 February 2014. Retrieved 21 February 2014.*
15. ^ *"Distributing your app to registered devices". Apple Developer Documentation. Archived from the original on 2024-03-13. Retrieved 2024-01-15.*
16. ^ *"Fake antivirus solutions increasingly have stolen code-signing certificates". 9 January 2014. Archived from the original on 16 April 2014. Retrieved 14 April 2014.*
17. ^ *"Why Private Keys Are the Achilles' Heel of Code Signing Security".*
18. ^ *".NET Security Blog". learn.microsoft.com. August 6, 2021. Archived from the original on January 19, 2024. Retrieved March 13, 2024.*
19. ^ *"Baseline Requirements for the Issuance and Management of Publicly-Trusted Code Signing Certificates" (PDF). CA/Browser Forum. 2024. p. 10. Archived (PDF) from the original on March 13, 2024. Retrieved March 22, 2024. "(Section 1.2.2) [...] Effective June 1, 2023, for Code Signing Certificates, CAs SHALL ensure that the Subscriber's Private Key is generated, stored, and used in a suitable Hardware Crypto Module that meets or exceeds the requirements specified in section 6.2.7.4.1 using one of the methods in 6.2.7.4.2."*
20. ^ *"Code Signing - Storage of Private Keys | SignPath". SignPath - Code Signing Simple and Secure. Archived from the original on 2024-03-08. Retrieved 2024-03-13.*
21. ^ *"Code signing changes in 2021". knowledge.digicert.com. Archived from the original on 2023-12-10. Retrieved 2024-03-13.*
22. ^ *"DigiCert timeline: Code signing's new private key storage requirement". knowledge.digicert.com. Archived from the original on 2023-12-08. Retrieved 2024-03-13.*
23. ^ *"New private key storage requirement for Code Signing certificates". knowledge.digicert.com. Archived from the original on 2024-02-19. Retrieved 2024-03-13.*
24. ^ *"[CSCWG-public] Voting results Ballot CSCWG-17: Subscriber Private Key Extension". 26 September 2022. Archived from the original on 2022-12-05. Retrieved 2024-03-13.*

25. ^ *"Code Signing Key Storage Requirements Will Change on June 1, 2023"*. *Archived from the original on October 2, 2023. Retrieved March 13, 2024.*
26. ^ *"460 Day Code Signing Certificate Validity (Update)"*. *SignMyCode. 7 February 2026. Archived from the original on 7 February 2026. Retrieved February 16, 2026.*

- Apple Code Signing Guide
- Microsoft Introduction to Code Signing
- Debian Security Infrastructure
- Strong Distribution HOWTO
